The WinFS API: Managed Access to WinFS Data

Mike Deem
Lead Program Manager
WinFS API Team

WinFS

Agenda

- API goals and design considerations
- Architecture drilldown
  - Data and framework classes
  - Object identity map and change tracking
  - Transactions, versioning, and behaviors
  - Watchers and rich application views

- This is not a WinFS API tutorial
  - See http://winfsapi for getting started material
  - I will show some code at the end if there is time

WinFS Item Store

- Complete data sub-system
  - Object, relational, binary, and XML data
- Schema framework
  - Extensible Windows schemas
  - ISV-built schemas
- Data model
  - Items
  - Relationships
  - Extensions
- Data services
  - Synchronization
  - Rules
  - Notifications
  - File system
- NTFS
  - NTFS semantics for file streams
  - Backwards compatible with Win32

WinFS API Goals

- Directly support WinFS core concepts
  - Items, extensions and relationships
  - Security model and sharing between users
  - Organization and operation models
  - Synchronization, proxy items, rules, and change notification
  - Data sharing across applications and type reuse (scenario extensibility)
  - Complex behaviors associated with types

- Provide great "out of the box" experience for developers
  - Developers can leverage familiar framework patterns
  - All WinFS types are a first class part of WinFX
  - Consistent programming model across diverse types
  - Intellisense and the object browser can be used for discovery
  - No need to learn prerequisite APIs

Complexity in the WinFS Environment

- CLR types, SQL types, and WinFS types
  - The store uses SQL types while the WinFS API is CLR type centric
- Local and Remote
  - Programming model must work well in both local and remote scenarios
- Two Tier and N-Tier (Longhorn Server Timeframe)
  - API will need to run in the client, mid-tier, and backend
  - API will need to support remoting for interaction between tiers
  - Applications written today have to keep working
- Tables and Items (Longhorn Server Timeframe)
  - The WinFS vision positions it as *the* store for *all* data
- File System and Database
  - WinFS is an item store, but must provide a first class support for files
  - WinFS supports stream properties for efficient I/O
  - WinFS is both, but traditionally the APIs are very different

File System and Database APIs

File System API Characteristics

- Basic pattern: open/lock, read, write, read, write close
- Operate on single file at a time, application coordinates activity
- Navigation over simple hierarchal organizational structure
- Immediate update of store on create, delete, rename, write, etc.
- Dumb backend, application has almost all the intelligence
- Data is opaque to the system, strong typing is up to the application

Database API Characteristics

- Basic pattern: query, modify, submit
- Operate on multiple rows at a time, system provides transactions
- Navigation over arbitrarily complex join based organizational structures
- Delayed update, waits for submit and transaction commit
- Smart backend, intelligence shared between application and store
- Data is schematized, system can provide strongly typed APIs

WinFS API Characteristics

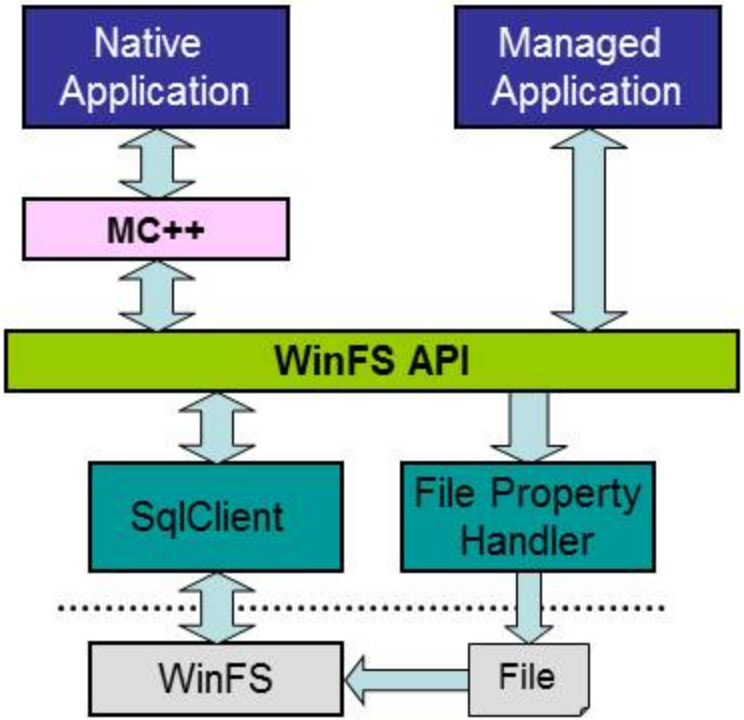
- The WinFS API follows the database API pattern
  - WinFS data model is fine grained, more like rows then files
    - One file may map to many items, relationships, and extensions
  - Changes may span multiple items, relationships and extensions
  - Batched operations work better in n-tier and remote scenarios
  - Intelligence can exist in the API, mid-tier, or backend
  - Relationships allow for complex structures
- Strongly typed objects based on WinFS schema
  - Not tables and rows
  - Query language (OPath) is over object graph, not SQL
- File Backed Items and Stream Properties
  - WinFS provides a file system redirector and can store files
  - WinFS copies selected file data into items for query und update
  - WinFS types can also have properties of type "stream"
  - WinFS API provides efficient I/O via System.IO.Stream
  - These features present some interesting challenges to the API

Additional Design Considerations

- WinFS is new, developers will have some things to learn
- The API should not hide core WinFS features
  - Might make initial experience a little easier
  - Would make developing an *accurate* mental picture harder
  - Doesn't mean the API shouldn't help the developer learn
- We need a single complete and simple API
  - Developers cannot be forced to drop down to SQL APIs
    - Too complex and too different
  - Providing distinct lower and higher level APIs is problematic
    - WinFS API objects are exposed by the Contact Picker, Shell, etc.
    - Layer of abstraction on WinFS may limit ISV adoption of platform
    - Many more types to test, document, support, etc.
  - Complex features cannot impact simple tasks

WinFS API Architecture

- All managed API
- MC++ for native apps
- Uses SqlClient for WinFS access
- Uses 3rd party file property handlers to update file backed items
- Runs entirely in application process
- WinFS process is the security boundary

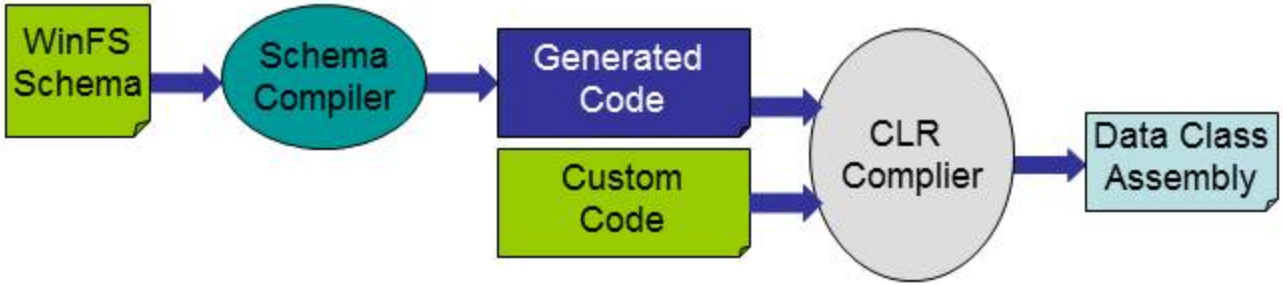


Overview of WinFS API Classes

- Data Classes
  - Represent data stored in WinFS
  - Strongly typed, generated from types defined in a WinFS schema
  - Examples: Person, Album, Document, PostalAddress
- Framework Classes
  - Supports API runtime operation
  - Examples: ItemContext, ItemSearcher, ItemWatcher, ItemView
- Design Time Classes
  - Generate data classes from schemas
- Management Classes
  - API for managing WinFS stores and shares
  - API for installing and uninstalling schemas
  - API for configuring synchronization and rules

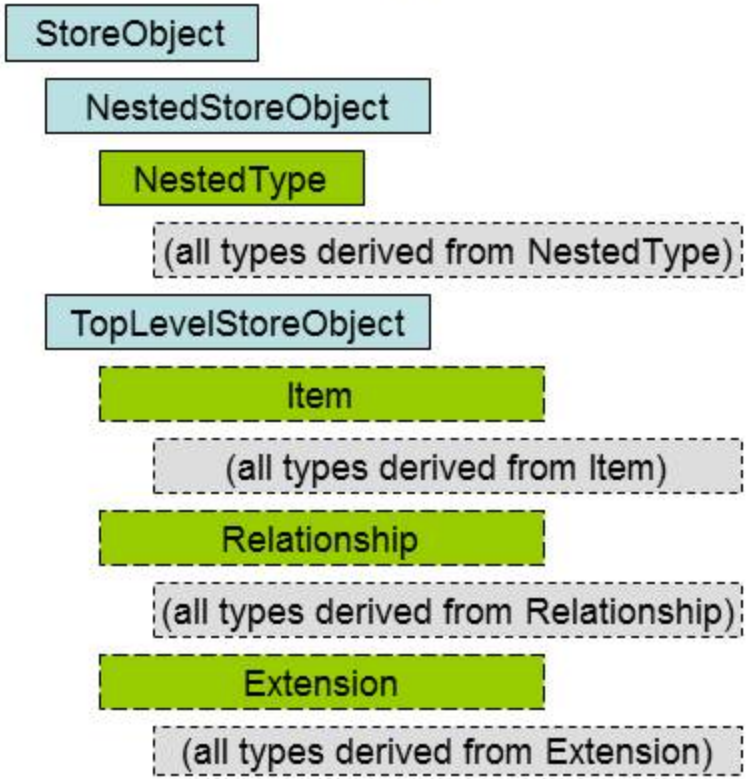
Data Class Creation

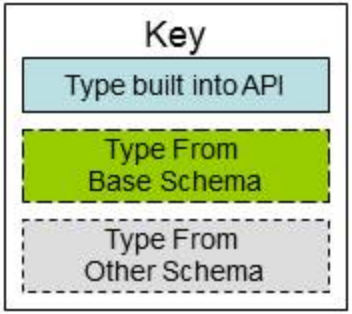
- Why generate classes?
  - Consistency: developer knowledge transfers across domains
  - Tax: bulk of class is rote code, no need to write by hand
  - Practical: there are lots of WinFS types, schemas change



- API Generator creates data class source code from schema
  - One class for each item, relationship, extension and nested type
  - One property for each property in the schema
  - Some additional methods and properties
- Type designer can augment generated code using partial classes

Data Class Type Hierarchy





Standard Data Class Features

- Avalon Data Binding
  - Winforms and Webforms to the point they align with Avalon
- Property Change Events
- Change Tracking
  - Report changes to ItemContext, ItemContext optimizes submit
- Standard Methods
  - GetSearcher, GetWatcher, etc.
- XML Serialization

Data Class Customization and Behaviors

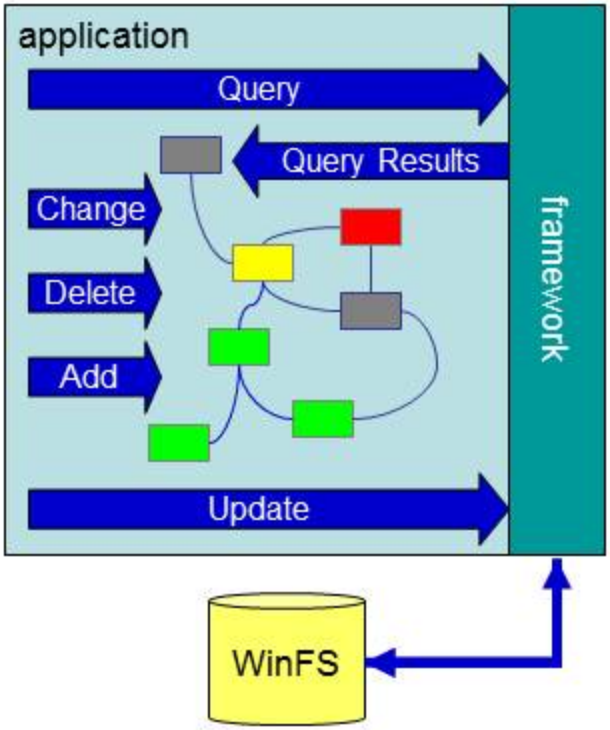
- Methods and properties can be added in partial classes
- API code generation can be customized via attributes in schema
  - Read only and internal properties
  - Read only classes with internal constructors
- Protected virtual "hook" methods can be overridden in partial class
  - OnAddRelationship, OnRemoveRelationship, OnRemoveExtension, OnLoad, other hooks for copy/move/serialize/deserialize are possible
  - A non-virtual pattern for hooking construction is also supported
- A pattern for "bindable" behaviors is supported
  - Classes implement hook methods and call methods on 3rd party class
  - The 3rd party object can be persisted as the value of a property

Framework Classes

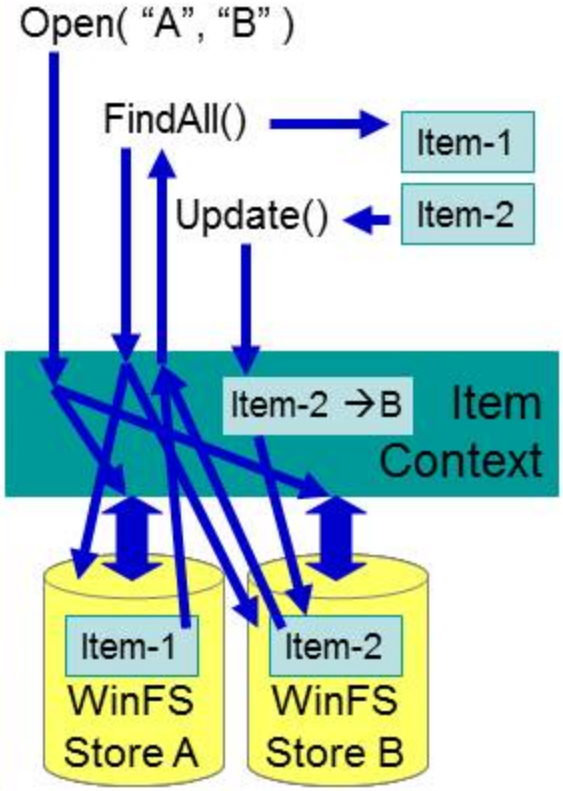
- ItemContext
  - Establishes and manages database connections with WinFS stores
  - Maps object query to SQL and constructs objects from results
  - Preserves object identity across queries
  - Tracks changes made to objects by the application
  - Manages transactions between WinFS and file system
  - Handles data class and schema versioning issues
- ItemSearcher
  - Encapsulates the data that describes a query
- ItemWatcher
  - Provides "watermarked" asynchronous change notification
- ItemView
  - Enables high performance and feature rich "live" views of data

ItemContext: The Find, Modify, Save Cycle

1. Application submits a find request to the framework
2. Framework maps request to WinFS query
3. WinFS returns the requested data
4. Framework returns item objects containing the requested data
5. Application changes, deletes, and inserts item objects
6. On save, changes to objects are mapped to changes in WinFS

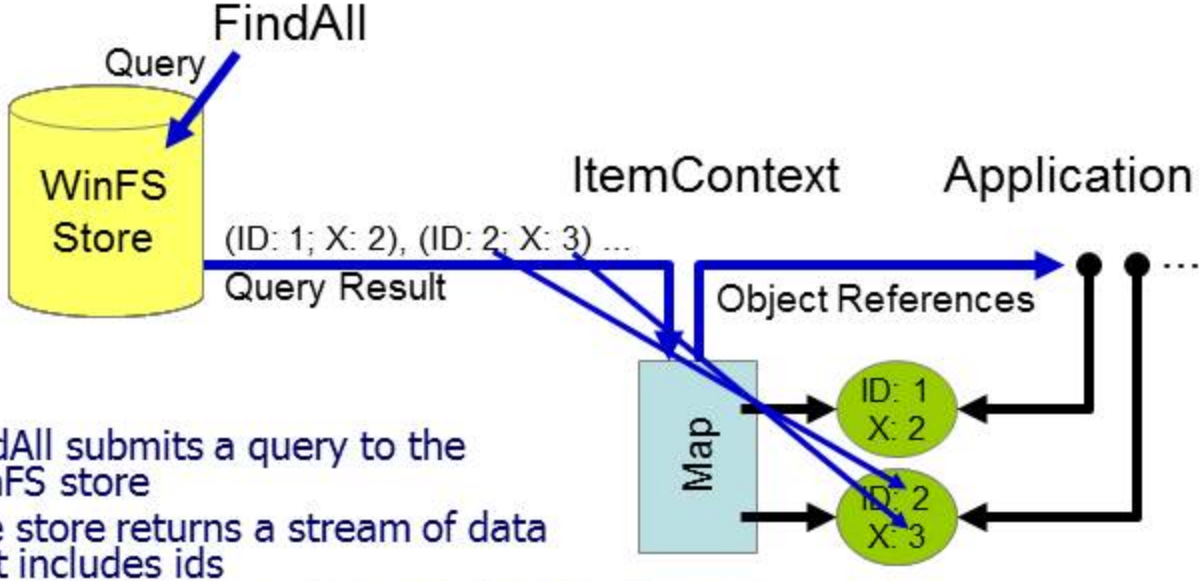


ItemContext: Connecting to Multiple Stores



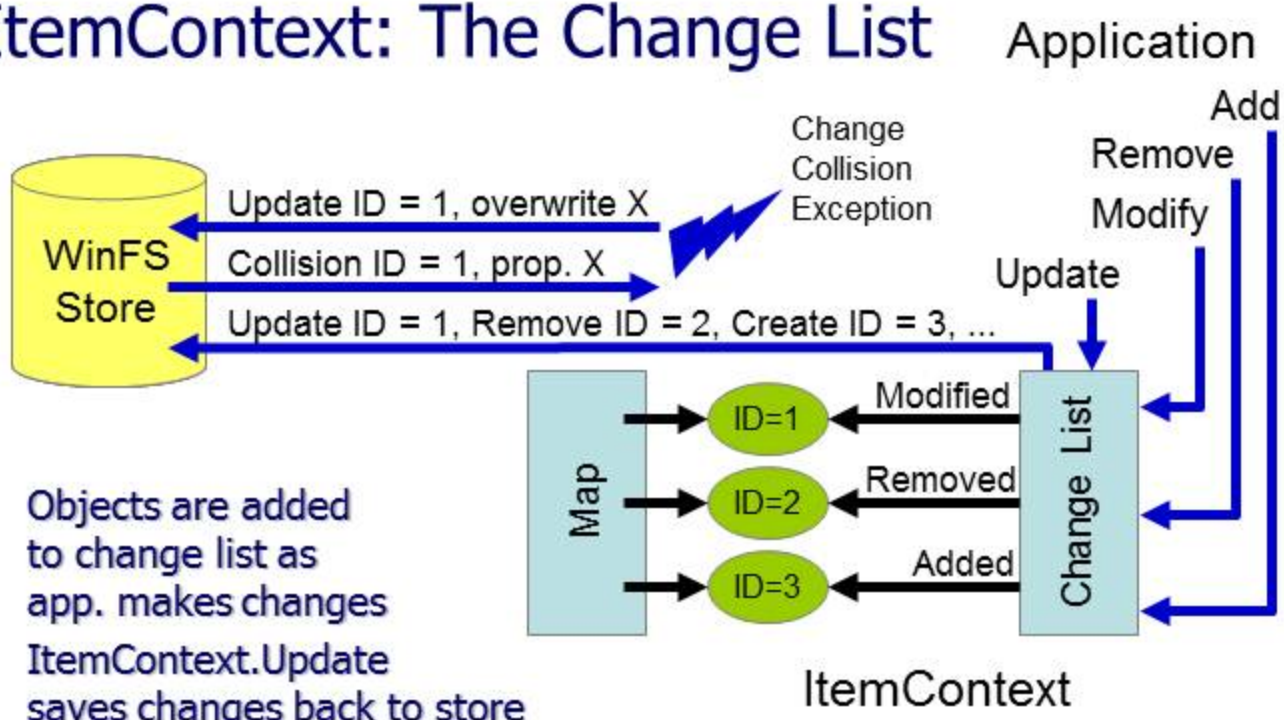
- Application opens multiple WinFS stores
  - Local or Remote
- Application submits query, ItemContext executes against each store
- ItemContext merges query results
- ItemContext remembers where object came from so it can be updated

ItemContext: The Object Identity Map



- FindAll submits a query to the WinFS store
- The store returns a stream of data that includes ids
- New objects are created and added to the map
- Objects already in the map are found
  - Object is updated with latest data from the store
  - Changes made by the application are not overwritten
- References are returned to the application

ItemContext: The Change List



- Objects are added to change list as app. makes changes
- ItemContext.Update saves changes back to store
- If another application has modified data a "change collision" exception is thrown
  - Application can choose to overwrite change in store

ItemContext: Transactions

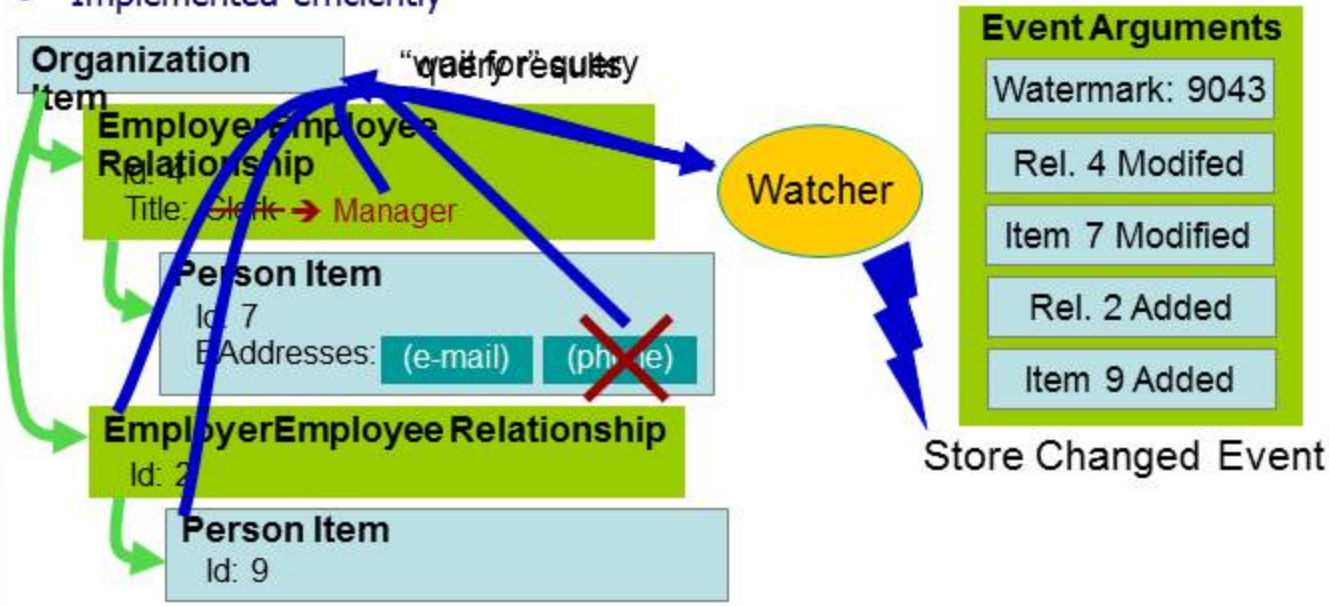
- Implicit Transactions
  - ItemContext automatically uses transactions to group updates
  - Uses the "read committed snapshot" isolation level
  - Each connection is updated independently
  - Exceptions and conflict resolution happen outside the transaction
- Transactions, file stream properties, and file backed items
  - File streams in WinFS use the new NTFS transaction support
  - Opening a stream can only be done inside a transaction
  - If there is no explicit transaction, the API starts one for each open
  - By default changes to streams and other properties are independent
  - Modifying a file backed item causes the backing file stream to be opened
- Explicit Transactions
  - Primarily useful to coordinate updates to streams and item data
  - Only the "read committed snapshot" isolation level is supported
  - The application has to clean up objects when a transaction fails

ItemContext: Versioning

- Versioning in the context of WinFS means:
  - Designing and installing new versions of schema
    - Like CLR, we support platform and library schemas
    - Platform places limits on allowed changes to schemas
    - One version of a platform schema can be installed at a time
    - Library schemas allow side-by-side storage, but limit sharing
  - Querying down level stores from up level clients
    - An application should not have to know what version the store has
  - Storing up level data in down level stores
  - Handling up level query results in down level clients
- It slices it dices: the new and improved Store-O-Matic®
  - The store and the API can "strip off" up level data
    - Unknown properties get "sliced" out of an instance
    - Unknown types get "diced" down to a known base type
  - Behavior properties are special: slice/dice makes class read only

ItemWatcher: Monitoring the Store for Changes

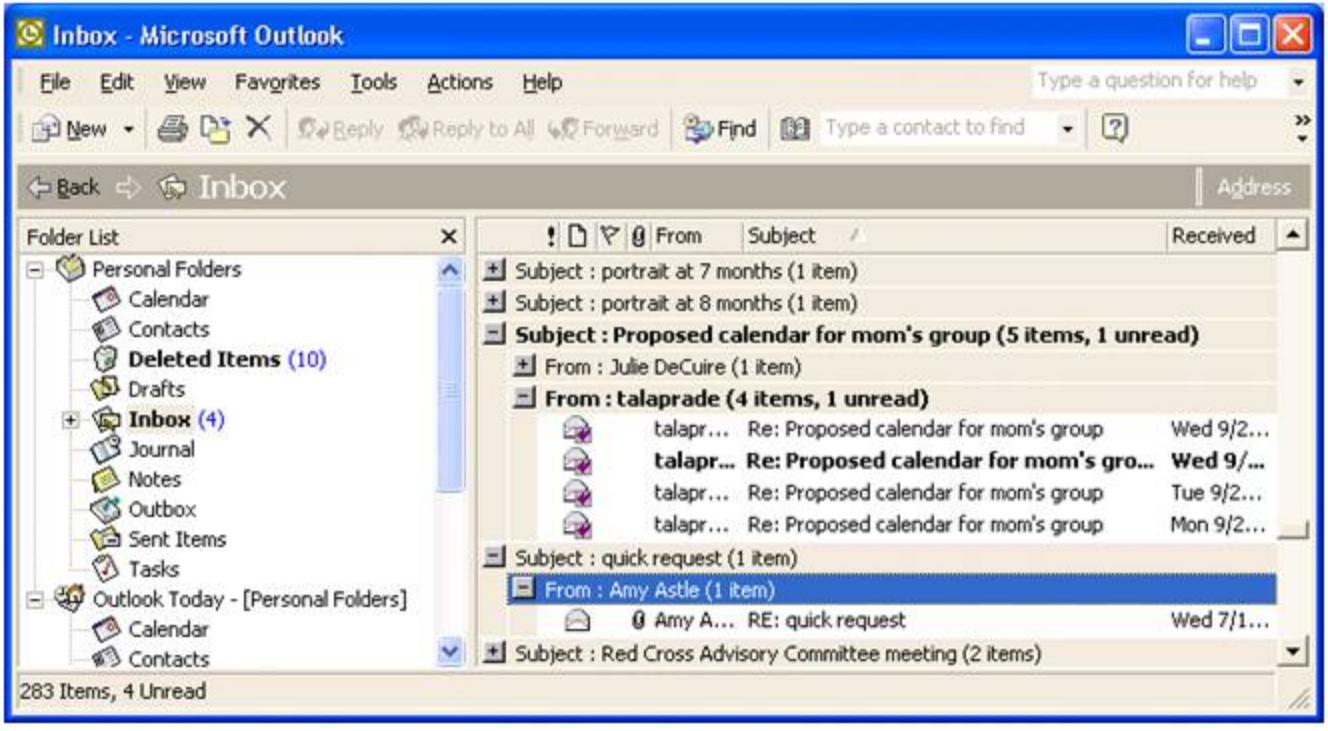
- Asynchronous notification of changes in the store
- Watch item, immediate children, or all children
- Filter by type of item, relationship, or extension
- Can save state and resume from same point
- Implemented efficiently



ItemView: Rich Application Views (RAV)

- RAV provides feature rich access to WinFS data
  - Data Virtualization and Paging
    - Enables viewing of very large data sets efficiently
  - Grouping, Sorting, Filtering and Aggregation
    - Enables creation of rich UI like Outlook Conversation view
  - Notifications
    - Application automatically notified as data changes in store
  - Data Shaping
    - Enables fine-grained control of exact data returned
    - Can project scalar, object, and collection values
- RAV is not a UI control and it's not a SQL View
- Also supports fast one-way read of view data (Projecting) for reporting, etc.

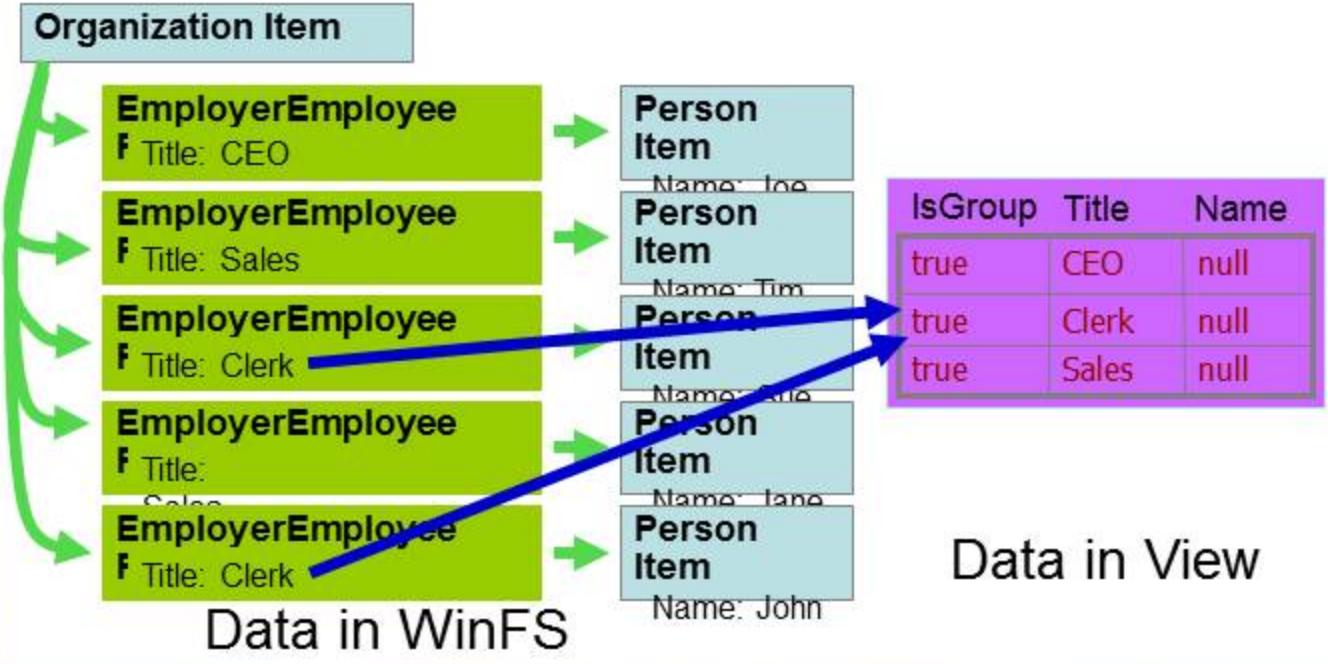
ItemView: Example Application



Outlook Conversation view: Group by Subject then Sender

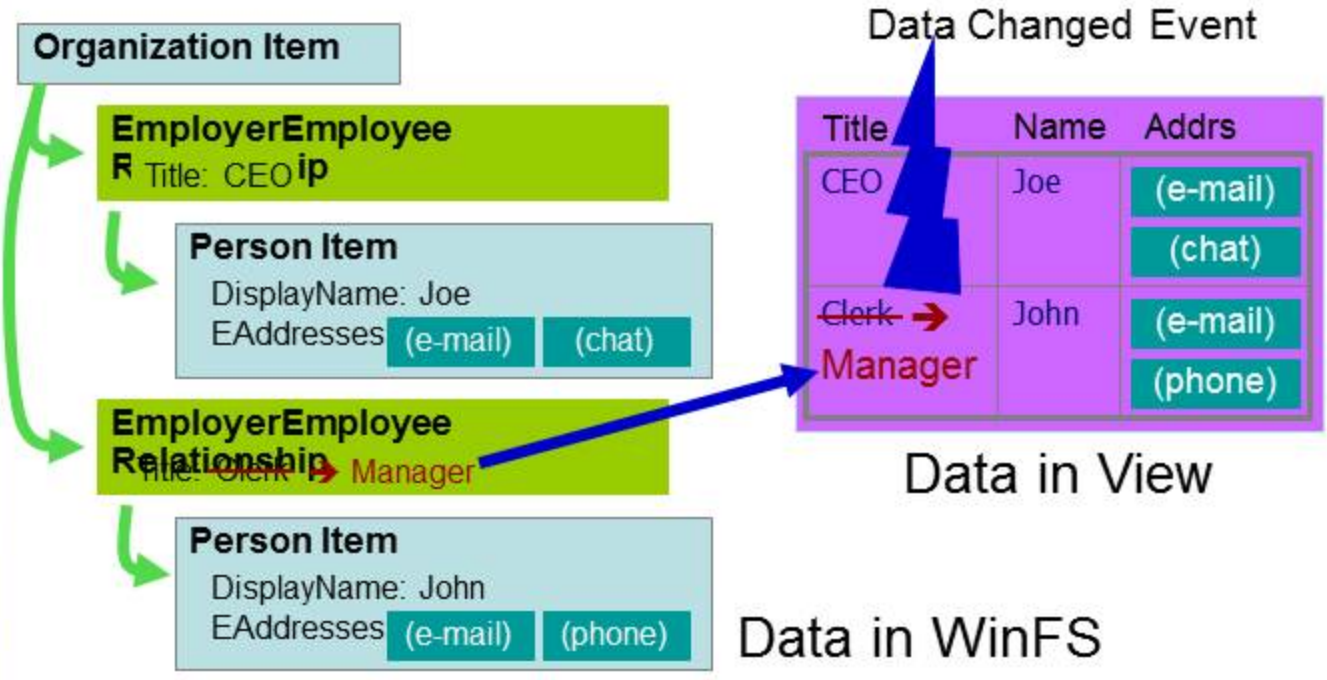
ItemView: Grouping

- Grouping by Title Adds Grouping Records to the view



ItemView: Change Notification

- Changes to data in the store are reflected in the view
- Applications are notified of changes in the view



What is Next?

- Adapt ItemSearcher to align with ISequence<T>
- Address ObjectSpaces and MBF requirements
- API changes to improve usability
- API changes to track WinFS data model changes
- Continued performance improvements

Demo: Code Examples

- Did we save enough time?